

Nro. ord.	Apellido y nombre	L.U.

ORGANIZACIÓN DEL COMPUTADOR I - **Parcial**

Primer Cuatrimestre 2022

Ej.1	Ej.2	Ej.3	Ej.4	Ej.5	Nota

Corrector:

Aclaraciones

- Anote apellido, nombre, LU en *todos* los archivos entregados.
- El parcial es domiciliario y todos los ejercicios deben estar aprobados para que el parcial se considere aprobado. Hay dos fechas de entrega, en ambos casos el conjunto de ejercicios a entregar es el mismo. En la primera instancia deberán defender su trabajo frente a su tutorx, quien les ayudará también a encaminar el trabajo de los ejercicios pendientes, si los hubiera.
- El link de entrega es: <https://forms.gle/re4UQkWJXKCvtYMA9>. Ante cualquier problema pueden comunicarse con la lista docente o preferentemente con el/la corrector/a.
- La fecha límite de entrega es el domingo 26 de Junio a las 21:00. El coloquio será el jueves 7 de Julio de cursada de los jueves (TM: 9 a 13hs - TT: 17 a 21hs) de **forma presencial** en un aula que confirmaremos en breve.
- Todas las respuestas deben estar correctamente justificadas y de corresponder hacer una comparación con la máquina Orga1.
- El archivo a adjuntar puede ser .pdf o txt, el nombre del archivo debe cumplir con el siguiente formato `orga1_nombre_apellido_parcial.pdf` o `orga1_nombre_apellido_parcial.txt`.

Introducción

Este parcial está dividido en seis preguntas, las primeras cuatro se relacionan con la arquitectura RISC-V. Se trata de una especificación y no una implementación de un procesador concreto. Si bien la arquitectura es diferente a la que utilizamos en clase, encontrarán los mismos conceptos que estudiamos y trabajamos aplicados al diseño de esta ISA. Toda la información necesaria está disponible en la **Guía Práctica de RISC-V** que se puede acceder libremente en: <http://riscvbook.com/spanish/guia-practica-de-risc-v-1.0.5.pdf>. Les recomendamos que hagan primero una lectura completa de los primeros tres capítulos de la guía y luego intenten responder las preguntas. Los ejercicios varían en temática y complejidad así que también les recomendamos ordenar la resolución de los mismos como les resulte más sencillo. Las últimas dos preguntas se relacionan con extensiones que introdujimos a nuestra implementación de `Orga1SmallI`.

Ejercicios

Ejercicio 1 Sabiendo que `a1 = 0xffffffff`, ¿Cuánto queda almacenado en `a2` luego de realizar la operación: `andi a2,a1,0xf0f`?

Ejercicio 2 ¿Cómo se modifica el valor del registro (`x0`)? ¿Cómo maneja las escrituras a este registro y por qué lo hace de esa forma?

Ejercicio 3 ¿Cómo resuelve **BGT**(branch great than) con RISC-V?

Ejercicio 4 Queremos agregar un flag llamado `P` (paridad) que nos indica si el resultado de una operación en la ALU es par. ¿Qué cambios hacen falta hacer a `Orga1SmallI` para implementarlo? Descríbalos en detalle. ¿Cómo resolvería un salto condicional (`JP`) por paridad? Suponga que tiene espacio libre para la instrucciones en la memoria de la unidad de control y que puede agregar tantas señales a la misma como hagan falta.

La necesidad de realizar numerosas operaciones sobre datos estructurados en memoria parece seguir creciendo mientras que el costo y la escala de las operaciones realizadas por el procesador se vuelven más costosas en tiempo e infraestructura. Debido a esto se están evaluando alternativas para realizar diversas operaciones en memoria sin tener la obligación de mover datos entre los registros y la memoria principal.

Proponemos el uso de un controlador de memoria con las siguientes señales de control:

- **mem_op** es una señal de tres bits que indican la operación a realizar
 - 000 no realiza ninguna operación
 - 001 permite guardar el valor del registro de destino (**wrAddr**)
 - 010 permite guardar el valor del registro de fuente (**rdAddr**)
 - 011 realiza una escritura en la dirección de destino desde el valor encontrado en **inData** (equivale al **load** de nuestra arquitectura)
 - 100 suma el valor que se encuentra en la dirección de fuente al valor que se encuentra en la dirección de destino y lo guarda en la dirección de destino
 - 101 multiplica el valor que se encuentra en la dirección de fuente por el valor que se encuentra en la dirección de destino y lo guarda en la dirección de destino
 - 110 limpia el valor que se encuentra en la dirección de destino
 - 111 mueve el valor que se encuentra en la dirección de fuente (**rdAddr**) y lo copia en la dirección de destino (**wrAddr**)
- **enOut** permite volcar el valor encontrado en la dirección de fuente en el registro **outData**

Y las siguientes señales de datos:

- **wrAddr** indica la dirección de destino (la dirección de la palabra donde realizaremos una escritura)
- **rdAddr** indica la dirección de fuente (la dirección de la palabra donde realizaremos una lectura)
- **inData** indica el valor de la palabra a guardar en la dirección de destino
- **outData** indica el valor de la palabra a leer de la dirección de fuente

Ejercicio 5 Suponiendo que las señales de control están conectadas a la unidad de control (**mem_op**, **enOut**) y los registros de datos al bus (**wrAddr**, **rdAddr**, **inData**, **outData**):

Escriba un microprograma que permita mover el contenido del valor que se encuentra en la dirección referida por el registro indicado en el **operando Y del decoder** a su destino en la dirección referida el registro indicado en el **operando X del decoder**.